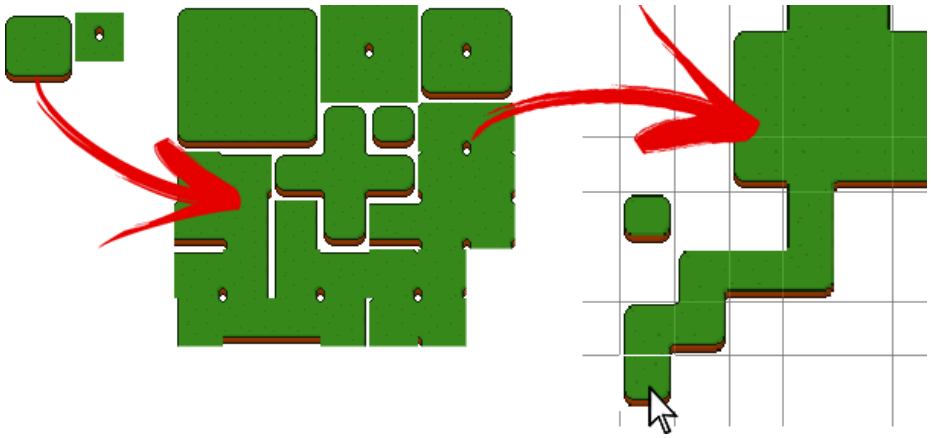
Tilegen Pro: Tileset Generator & Automatic Rule Tiles v0.97

By Igor Hatakeyama

If you have any questions or suggestions please contact me at igor.hatake@gmail.com

If you like the asset please leave a review on the asset store!

What does this asset do, exactly?



The asset works in two parts. First, it takes 13 tile quadrants as an input and compiles it into a bigger tileset containing many variations of tiles that the user would otherwise have to draw by hand. It does that by taking information already present in the base quadrants and creating new tiles using that information.
After the tileset is done, the algorithm automatically creates a rule tile using the tiles from your brand new tileset which can then be used to paint the tile freely on a grid tilemap.

As this tool uses rule tiles, you need to have the **2D extras** by Unity Technologies installed in your project, which can be acquired for free in the link provided under “Installation”, below.

Installation

There are different ways of getting the 2D extras package in order to make TileGen Pro work. Getting it from the official Unity Github repository and installing it as a package is confirmed to work. **If you don’t do this important step, you will get errors in your console, such as the “Missing RuleTile” error, and TileGen will not work.**

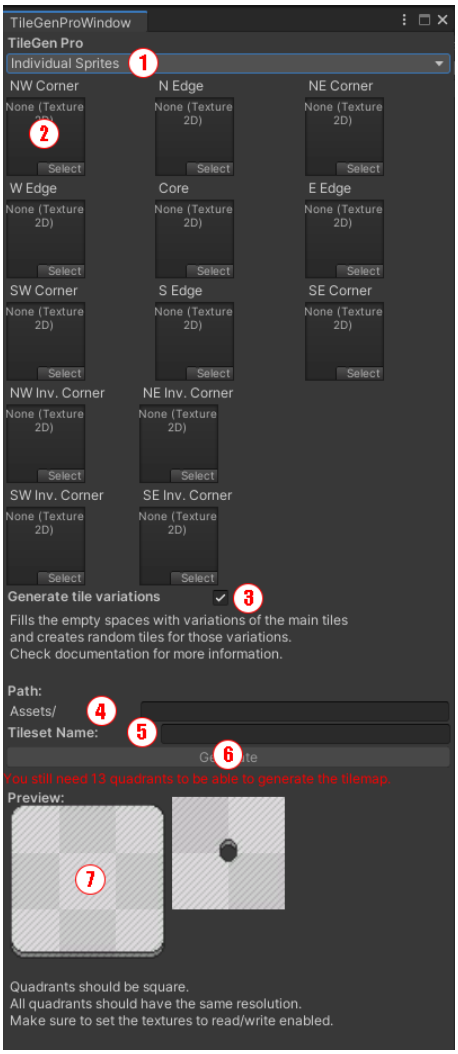
1. Open a new 2D unity project. It is confirmed to work with Unity 2019.4.29f1
2. Go to https://github.com/Unity-Technologies/2d-extras/tree/2019.4 and make sure you are on the 2019.4 branch

3. Download the repository to your machine and unpack the zip file anywhere
4. In unity, go to Window > Package Manager > Click on the plus sign on the top left side > Add package from disk…
5. Choose package.json under 2d-extras-2019.4\2d-extras-2019.4 in the repository you downloaded and import it.
6. You can now import TileGen Pro from the Unity Asset store

Getting started

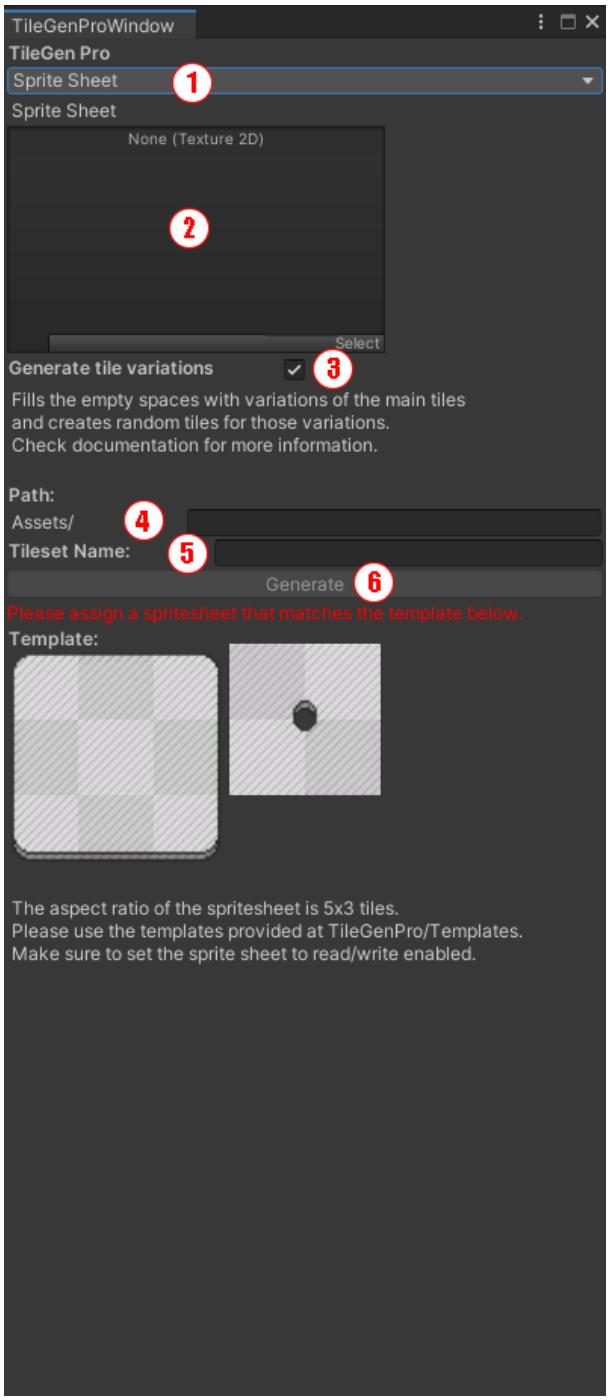
After you import TileGen Pro from the asset store, you can open the TileGen Pro window by going to Window/TileGen Pro. It should look like this:

This is Tilegen's window when it is configured for individual sprites.



1. **Input Mode**
   This is where you set the input type of be either individual sprites or a spritesheet. Both achieve the same results so it depends entirely on your preference.
2. **Texture input fields**
   This is where you input the 13 quadrants needed to create the tileset. The 13 quadrants need to be based on the same quadrants shown in the preview **(7)**
3. **Generate tile variations**
   This will make the tool generate additional sprites that fill up some of the empty space in the tilesheet. Select this if you want some of the more frequently used tiles, like the core tile, to have random variations.
4. **Path folder**
   Here is where you input the path to where you want your tileset to be. The format is "FolderA/FolderB". Please note that the "Assets" folder is automatically included in the path, so **you don't have to type it out.**
5. **Tileset name**
   This is the name of your tileset. If left blank, TileGen will generate a random name for you.
6. **Generate button**
   This is the button to press when all texture input fields are filled.
7. **Preview**
   This shows a preview of the quadrants needed as you add those quadrants as textures in the input texture fields.

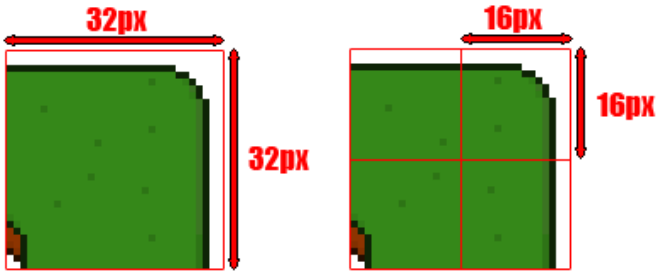
If you change the input mode to spritesheet, this is what the window will look like:



1. **Input Mode**
   This is where you set the input type of be either individual sprites or a spritesheet. Both achieve the same results so it depends entirely on your preference.
2. **Sprite sheet input field**
   This is where you input the sprite sheet containing the 13 quadrants needed. Note that the spritesheet needs to be in a specific format and aspect ratio.
3. **Generate tile variations**
   This will make the tool generate additional sprites that fill up some of the empty space in the tilesheet. Select this if you want some of the more frequently used tiles, like the core tile, to have random variations.
4. **Path folder**
   Here is where you input the path to where you want your tileset to be. The format is “FolderA/FolderB”. Please note that the “Assets” folder is automatically included in the path, so **you don’t have to type it out.**
5. **Tileset name**
   This is the name of your tileset. If left blank, TileGen will generate a random name for you.
6. **Generate button**
   This is the button to press when all texture input fields are filled.

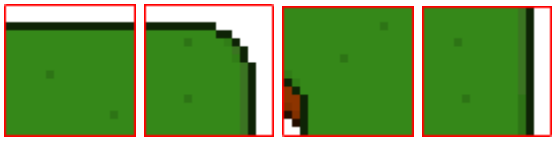
Quadrants

A quadrant is always a quarter of the resolution of your final tile. For instance, if your final desired tileset contains tiles that are 32x32, then a quadrant will be 16x16.



Shown on the left, a 32x32 pixel tile. Shown on the right, a 16x16 pixel quadrant within said tile.

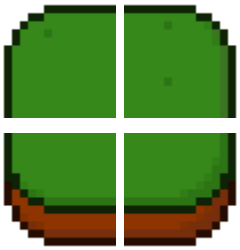
So we can say that this 32x32 pixel tile is made from these 4 16x16 pixel quadrants:



Quadrants Needed

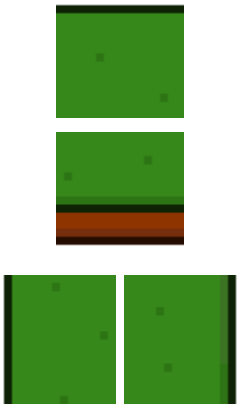
To use TileGen Pro, we need 13 quadrants:

4 Corner quadrants:



NW Corner, NE Corner, SW Corner, SE Corner

4 Edge Quadrants:

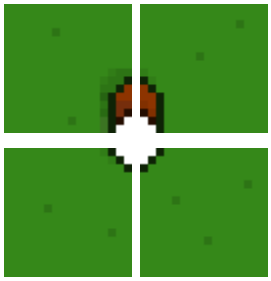


N Edge, S Edge, W Edge, E Edge

1 Core Quadrant:

Core

4 Inverse Corner Quadrants:

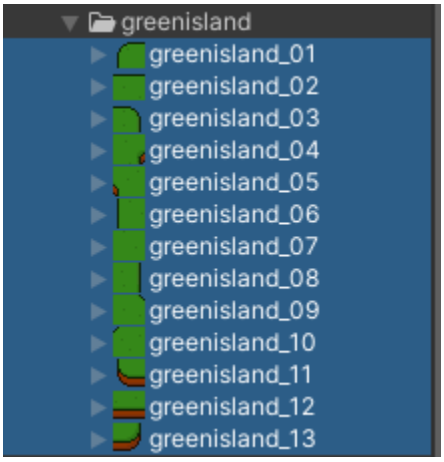


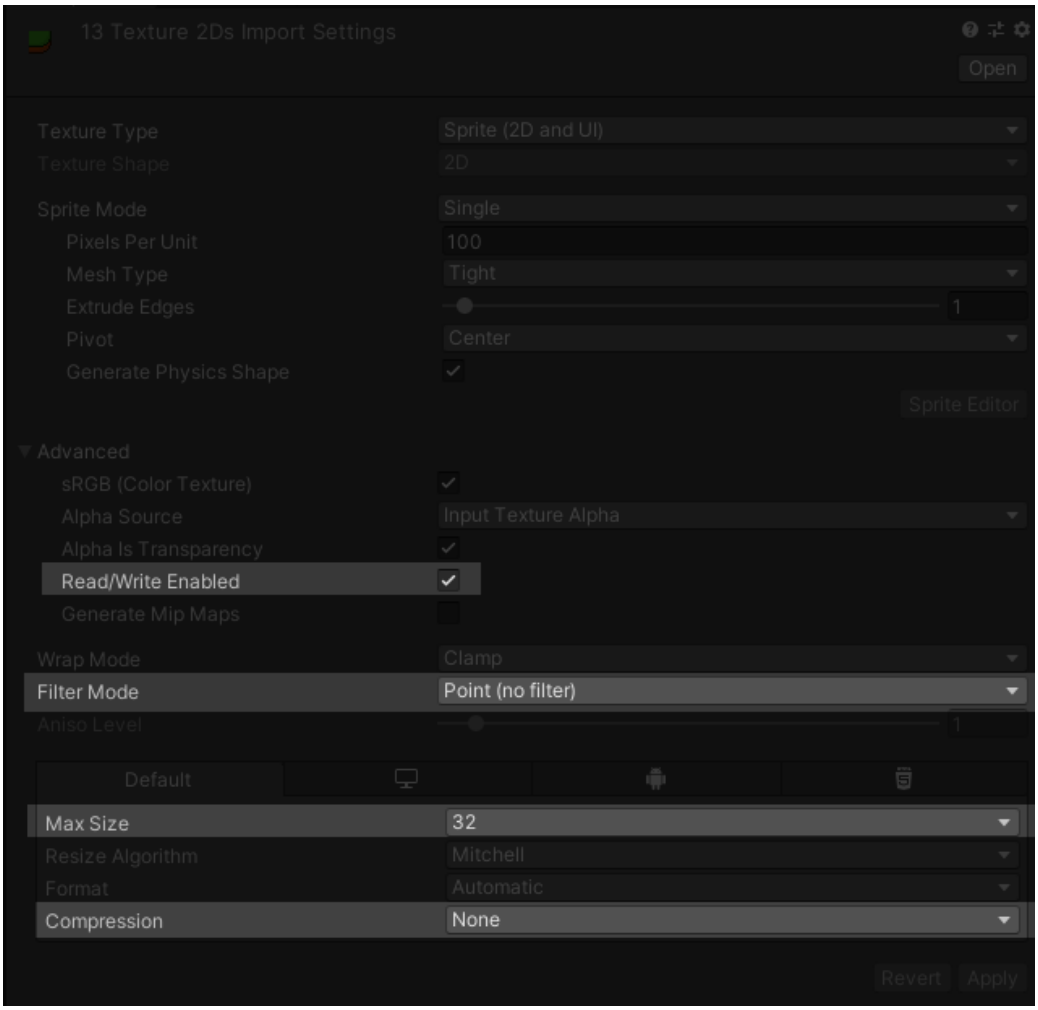
NW InvCorner, NE InvCorner, SW InvCorner, SE InvCorner

These 13 quadrants are needed in order to use TileGen Pro.

Configuring your Quadrants sprites

After you create your 13 quadrants needed to use Tilegen, there are a few things you still need to do. First, select all of your quadrants in order to edit them all at the same time.





On the import settings of the 13 quadrant textures, there are a few things you have to set up.

Read/Write Enable:

This should be set to true, otherwise TileGen Pro will not be able to read these textures to generate the tileset.

Filter Mode:

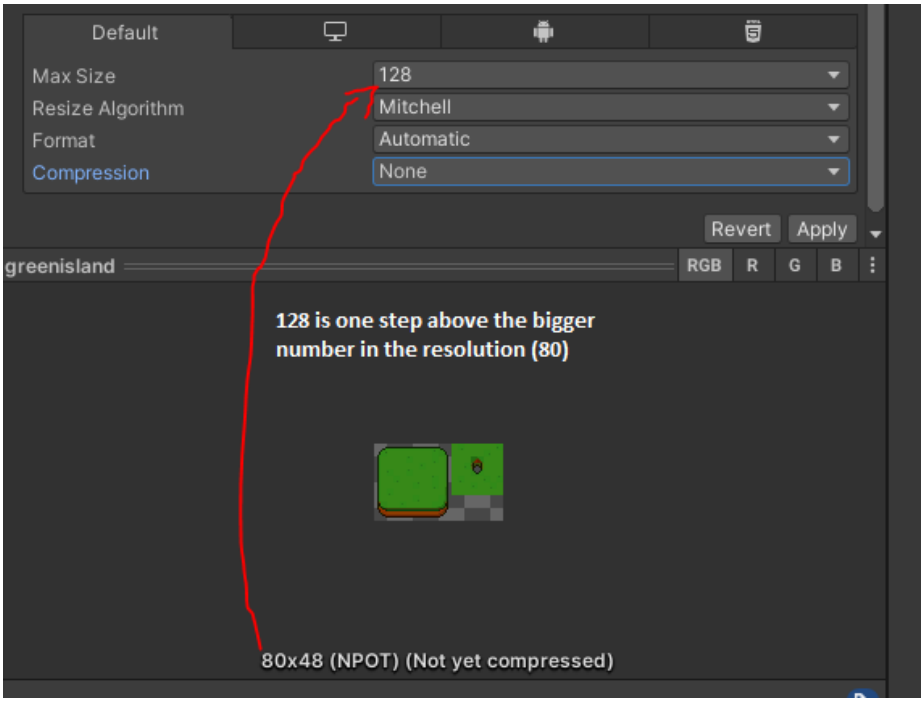
This should be set to Point (no filter) at lower resolutions if you want that crisp pixel art look.

Max Size:

This depends on whether you are using individual quadrants or a spritesheet.

Individual Quadrants: should be the same size as your sprite. In my case, the quadrants are 16x16 pixels, and since the lowest Max Size is 32, I set it to 32. If each quadrant was 32 I would still set it to 32, if each quadrant was 64, I would set it to 64, etc.

Spritesheet: Should be the same or bigger than the biggest resolution of the spritesheet. So for instance, tgp_template_16x16.png is 40x24 pixels. 40 is the bigger value, but also there is no max size option for 40 pixels, so you just pick the step above, in this case, 64. If your spritesheet was 80x48, you'd pick the option above 80 (the bigger number), which would be 128, and so on.



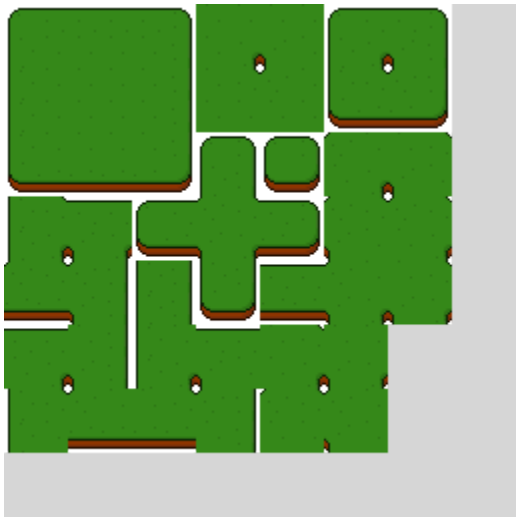
This may be difficult to understand, so in summary:
Take your spritesheet, check its resolution. Pick the bigger number of the resolution (so for instance, if the resolution is 20x12, you pick 20) and then you choose the max size option that is one step above that resolution (so in the case of 20, the next step above it is 32).

Compression:
Set it to none in order to preserve the quality of the image.

After this is done, you can open the TileGen Pro window and assign each sprite to its specific texture input field.

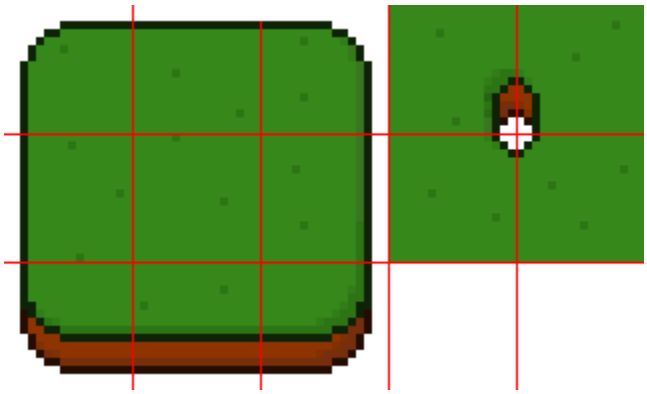
You can then press generate.
It should then create a tileset sprite and a rule tile in the folder that you set up.

This is the tileset created from it:

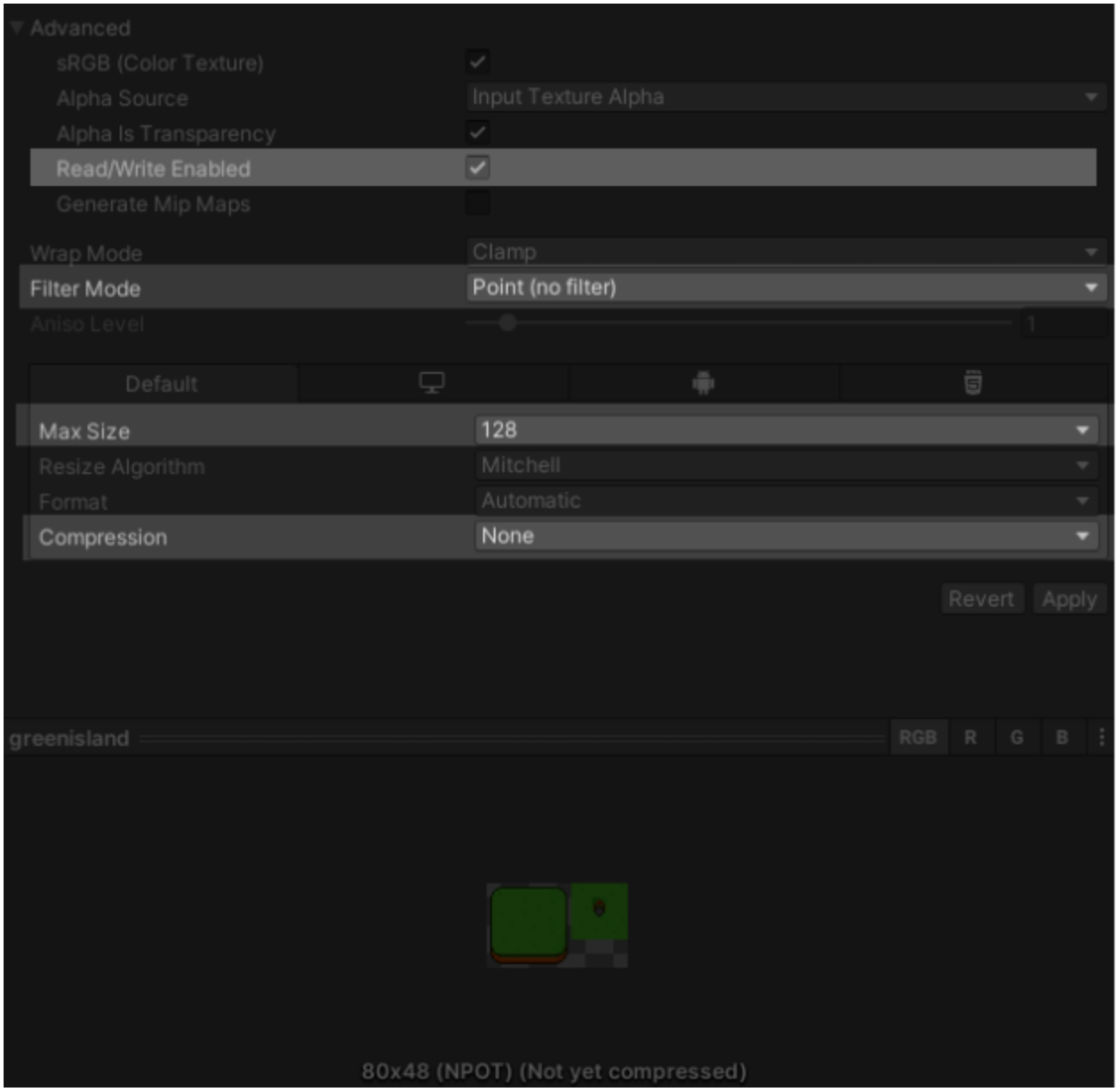


A rule tile is also created with it.

If you wanted to create this tileset using the spritesheet method (as opposed to the individual quadrants method), it's easy. Instead of exporting each quadrant as a spritesheet, export them following this format:



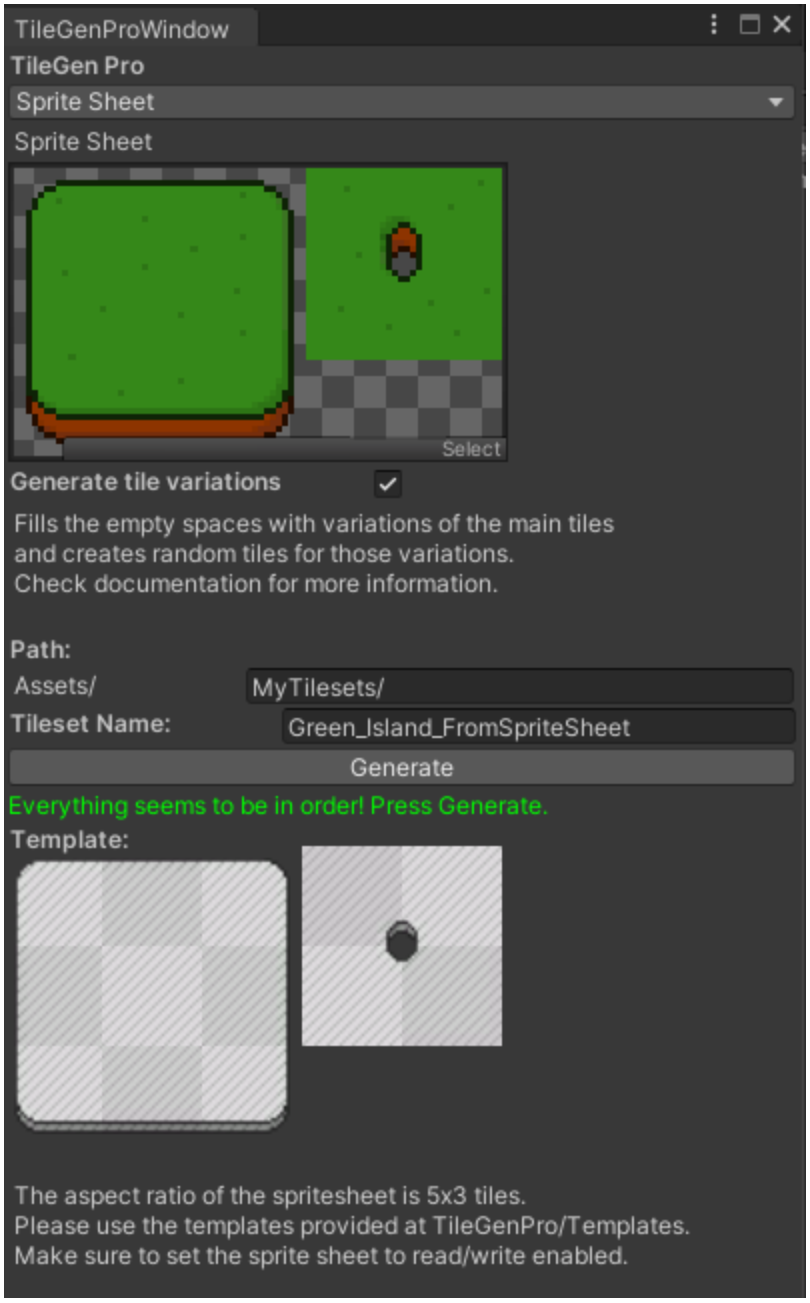
No matter your resolution, this tilesheet will always need to have an aspect ratio of 5 quadrants wide by 3 quadrants tal. There are always two empty quadrants at the bottom right that will not be used.



In the tilesheet's texture import settings, we should set **read/write enabled** to **true**, we should also set the **filter mode to point** (if we wish for our tileset to have a crisp pixel art look), the **compression to none**, and the **max size** needs to be **one step higher than the biggest dimension of the resolution of the tilesheet**.

So since our tilesheet is 80x48, 80 being the bigger number, we set the max size to 128, since 128 is the biggest step that comes after 80. If our spritesheet was bigger than 128 and smaller than 256, we'd set the max size to 256, etc.

After you set it up, you can hit apply, and set TileGen Pro to the sprite sheet input type. You can assign the sprite sheet to the input field and press generate.



As you can see, the tileset and the rule tile generate just the same.

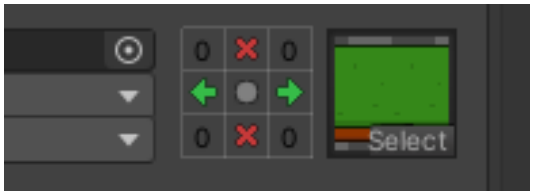
The Rule Tile

Unity's rule tile is an impressive asset that works perfectly with the complex tileset that TileGen creates. It works by letting the user define rules that the tile will follow, based on the neighboring

tiles around that tile. If a certain rule is met, a certain tile will be displayed. Rules are based on neighboring tiles - you can set a rule to be either:

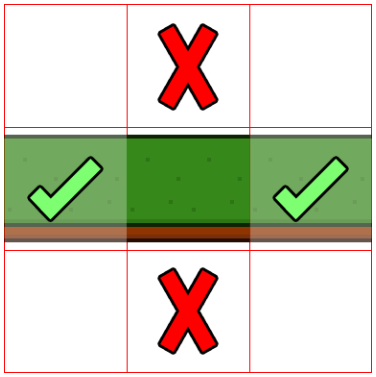
- “There needs to be a tile here” (**green** arrow)
- “There cannot be a tile here” (**red** X)
- “Irrelevant” (**nothing**).

Let’s take a closer look at the west/east bridge tile and it’s rules.

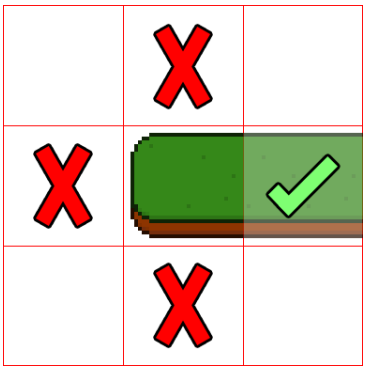


The neighboring tile rules translate to:
There **cannot** be any tile north and south of this tile
There **needs** to be tiles east and west.
It’s **irrelevant** whether there is a tile or not on the rest of the neighboring positions.

If we put this on a grid, we will see that it makes sense:



If we, for instance, make it so there is no tile on the west neighboring position, we create a west-facing end tile:



And if we go back to Unity and check the rule for the west-facing end tile, we’ll see that it looks like this:

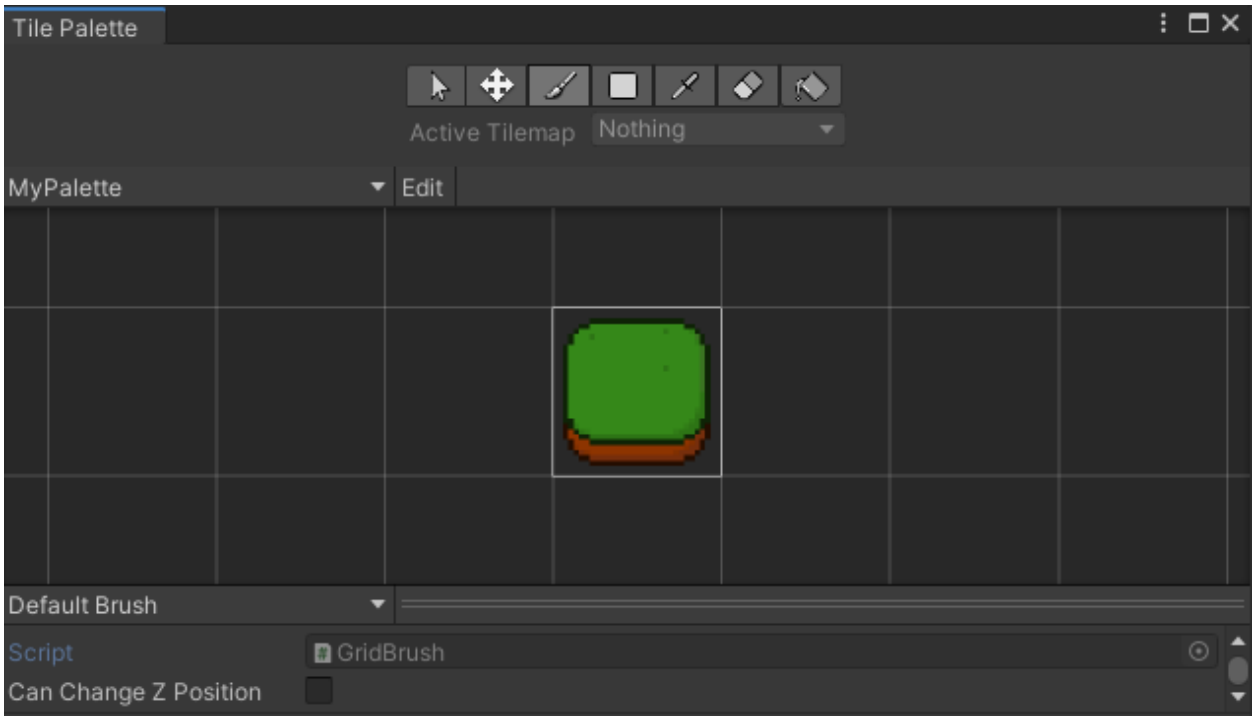
Which is exactly what we expect.

TileGen provides all of the complicated rules, set up in a way that some rules are prioritized over others, to avoid visual glitches while painting with the rule tile.

How can I paint using the Rule Tile?

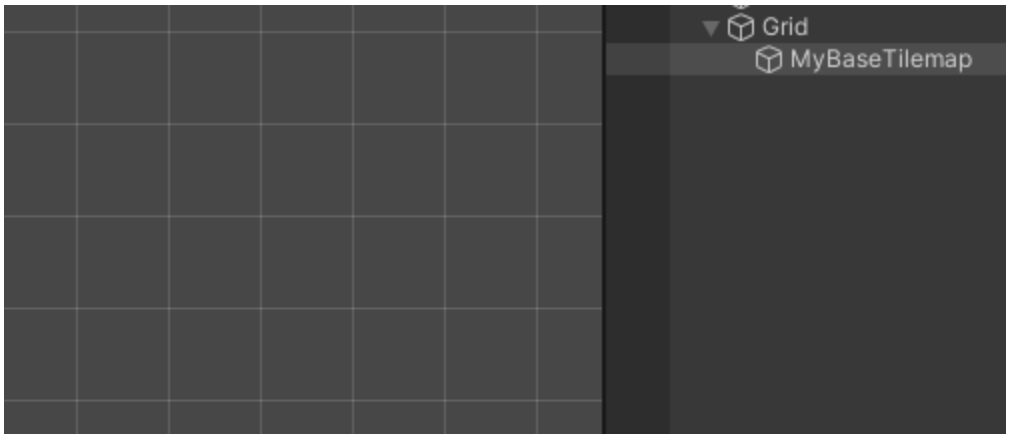
Once TileGen Pro generates a rule tile for you, simply click on Window > 2D > Tile Palette. This will show you your Tile Palette, where you can drag and drop tiles, rule tiles and sprites that you can use to paint with later on. Just click on Create New Palette, name it, and choose a location to keep it.

Then you can drag and drop your rule tile to the center of the tile palette area.

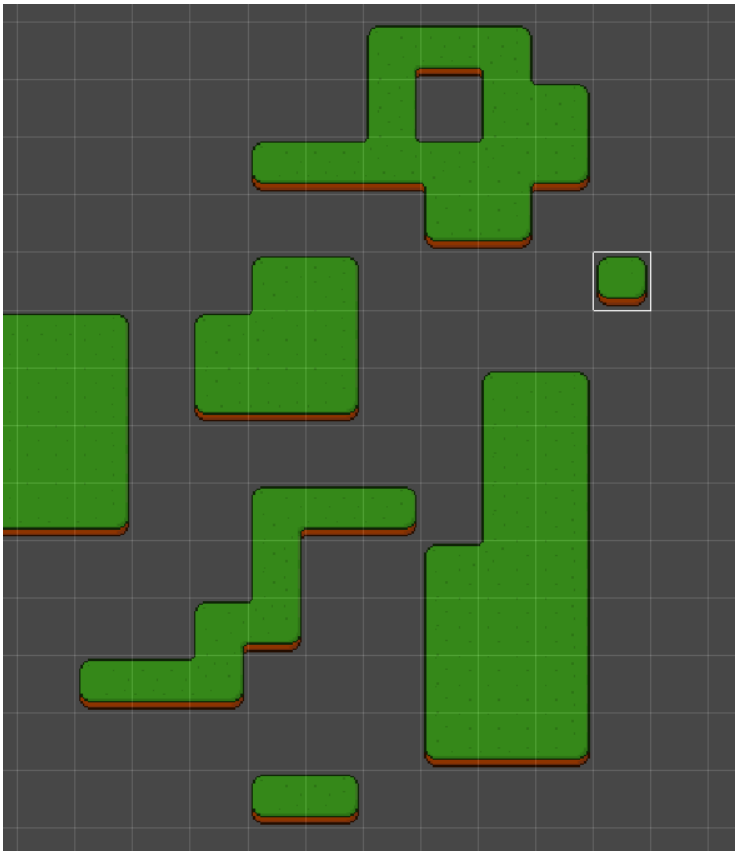


Now you need to set up a grid and a tilemap.

Go to Gameobject > 2D Object > Tilemap. If you didn't have a grid in your scene before, it will create a grid for you and will add a Tilemap as a child.

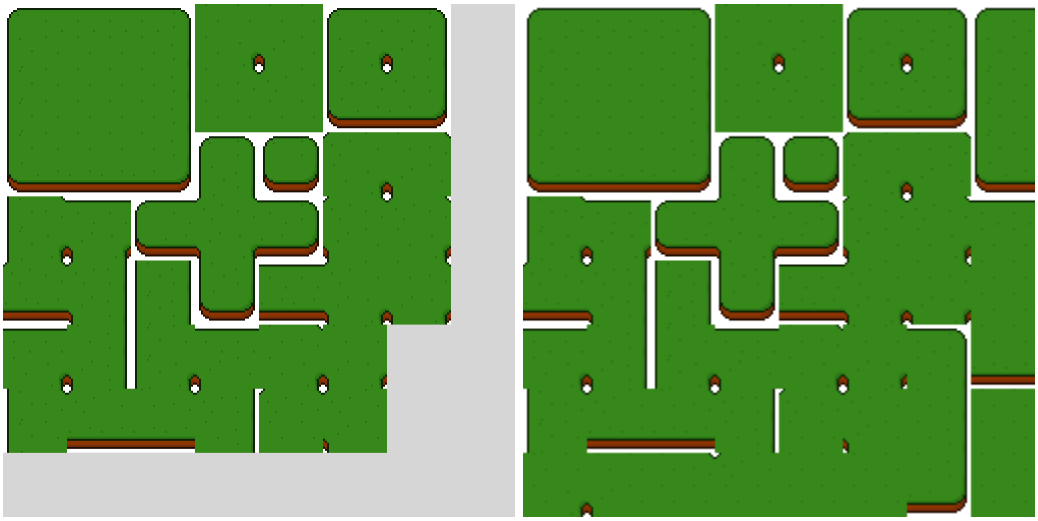


Now go back to your tile palette. Select the rule tile you just dragged into it. Select the tilemap you want to paint on under “Active Tilemap”. Select the brush tool, and now you can paint onto the tilemap in the scene.



Tile Variations

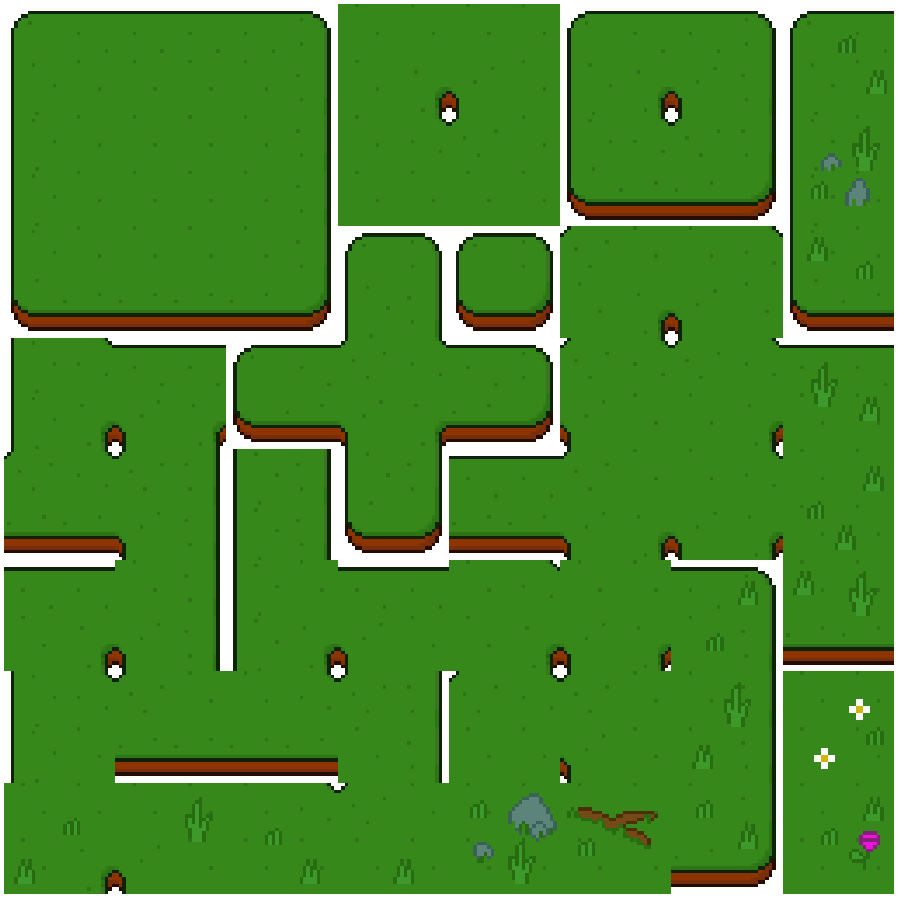
When generating your tileset, if you click on “Generate Tile Variations”, you will notice that your tileset will have some tiles that fill the empty spaces on the bottom right of the tilemap.



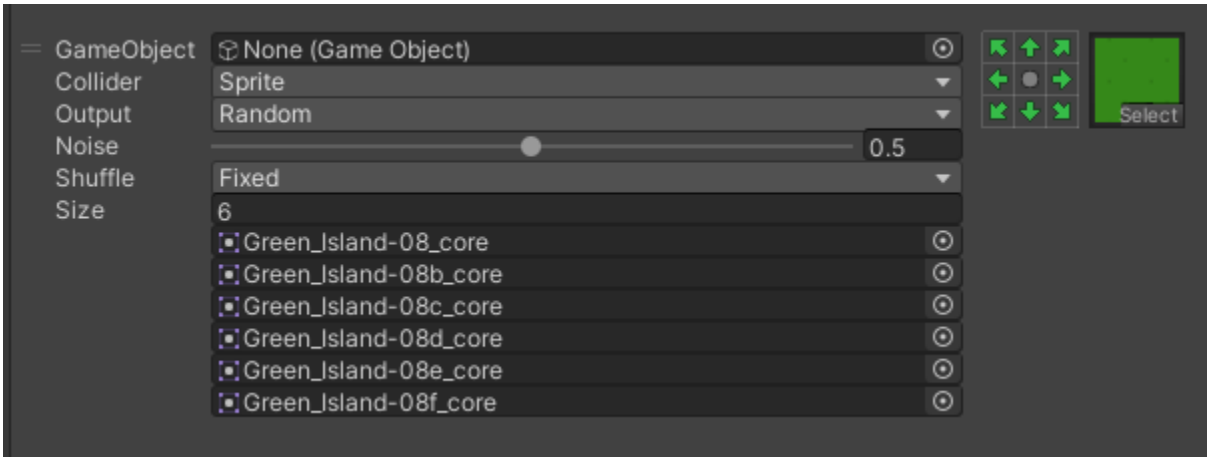
A tileset without tile variations on the left, and a tileset with tile variations on the right.

These tiles are variations of pre-existing tiles that will be randomly placed upon drawing using the tilemap.

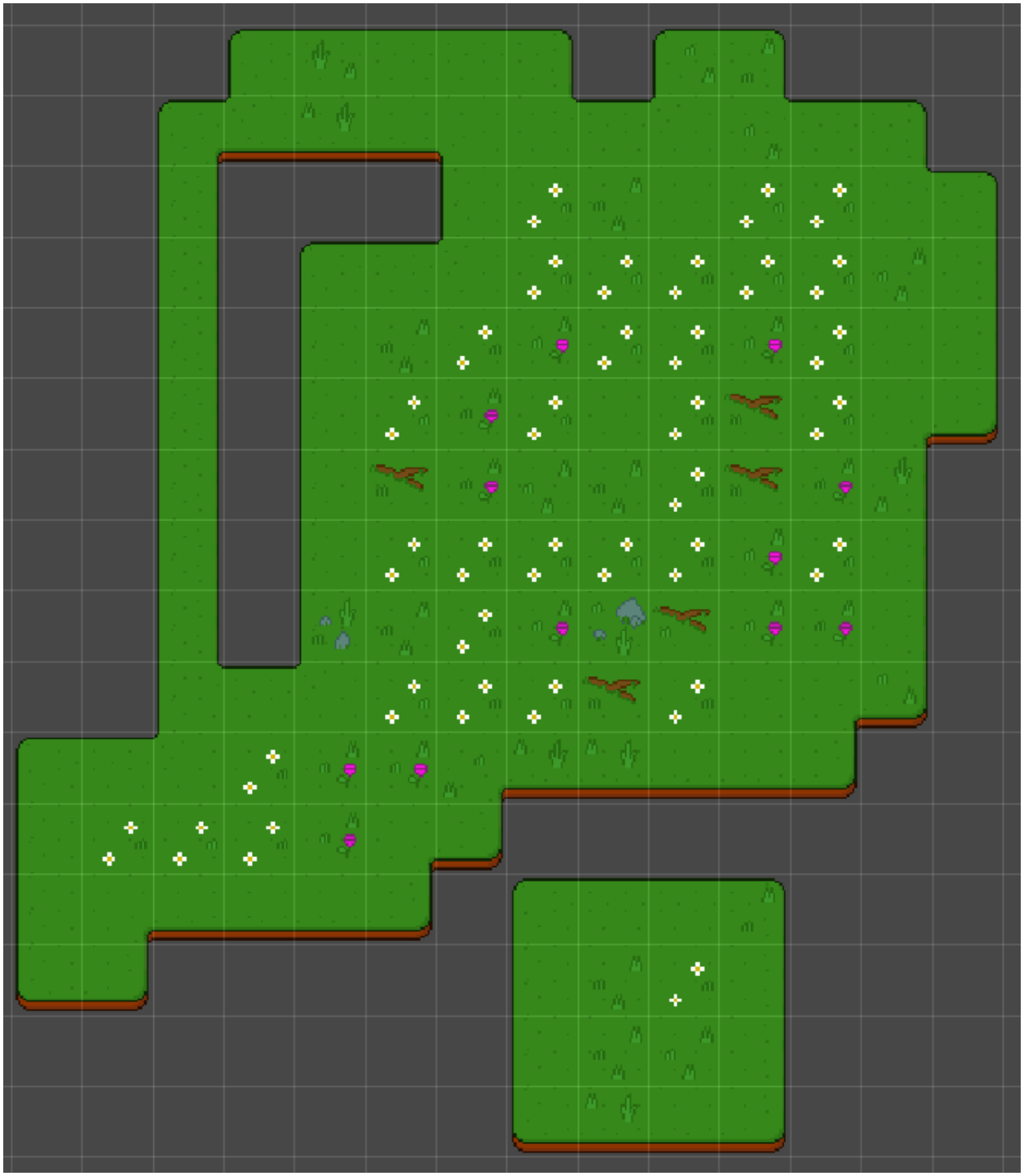
Each corner, edge and inverse corner has 1 variation. The core tile, since it is arguably one of the most used tiles, has 6 variations.



For this example, I edited the tilemap generated by TileGen Pro directly, and saved it with painted-over tile variations. I added a tall grass variation for each of the corner, edge and inverted corner variation tiles, and for each of the 6 core variations, I added things like flowers, branches, pebbles and rocks.



If you inspect the rules within the rule tile, you'll see that some rules, like the rule for the core tile, are set to Random output and have 6 sprites assigned to it. This means that when we draw with the rule tile, it will variate between the assigned sprites, which TileGen Pro assigned automatically. Remember that you can always add your custom sprites as variations to each rule. Let's see the effects of our change to the tilemap.



Now when we draw, the rule tile randomly places the variation tiles, when applicable.

FAQ and troubleshooting

Why is my tileset with 8x8 tiles not rendering properly?

This is possibly due to the fact that a spritesheet to make a tileset with 8x8 tiles is so small that it is way under unity's max size compression options, so it can look strange. A possible solution is to use individual tiles for quadrants instead of a spritesheet.

Why are there grey artifacts around my tiles?

This is most likely due to your tiles having black borders around them. There is some strange artifacting that happens when your tile has borders that are completely black (RGB 0,0,0). A workaround is making the borders slightly less dark. Local tests showed that even by making the borders 1% less dark removed the artifacting.

My tileset will generate, but the rule tile will come up empty and I won't be able to drag it to the tile palette.

This is an issue that can happen, especially if you are making larger tilesets. A workaround is to simply press "generate" again after you are done generating the problematic tileset, this forces the rule tile to generate properly.